Search Results

Search Results for: **[((1st or first or one or left*) <near/5> (hierarchy or hierarchies or tree* or subtree*)) <near/10> ((share* or common) <near/5> (node* or object or objects)) <near/10> ((2nd or second* or another or different or separate or adjacent or right*) <near/5> (hierarchy or hierarchies or tree* or subtree*))]**
Found **1** of **866,341 searched.**

Search within Results

Results 1 - 1 of 1 **short listing**

1 Planar point location using persistent search trees 95%

Neil Sarnak , Robert E. Tarjan
Communications of the ACM July 1986
Volume 29 Issue 7

A classical problem in computational geometry is the planar point location problem. This problem calls for preprocessing a polygonal subdivision of the plane defined by n line segments so that, given a sequence of points, the polygon containing each point can be determined quickly on-line. Several ways of solving this problem in O(log n) query time and O(n) space are known, but they are all rather complicated. We propose a simple O(log n)-query-time, O(n)-space solution, using persistent se ...

Results 1 - 1 of 1 **short listing**

Results for "((1st or first or one or left*) <near/5> (hierarchy or hierarchies or tree* or subtree*)) <..."

e-mail

Your search matched **1** of **1166705** documents.

A maximum of **100** results are displayed, **25** to a page, sorted by **Relevance** in **Descending** order.

View Session History

New Search

Modify Search

((1st or first or one or left*) <near/5> (hierarchy or hierarchies or tree* or subtree*)) <r

Key

IEEE JNL	IEEE Journal or Magazine
IEE JNL	IEE Journal or Magazine
IEEE CNF	IEEE Conference Proceeding
IEE CNF	IEE Conference Proceeding
IEEE STD	IEEE Standard

Check to search only within this results set

Display Format: Citation Citation & Abstract

1. **New results on the size of tries**
   Regnier, M.; Jacquet, P.;
   Information Theory, IEEE Transactions on
   Volume 35, Issue 1, Jan. 1989 Page(s):203 - 205
   AbstractPlus | Full Text: PDF(240 KB) IEEE JNL

```
File   8:Ei Compendex(R) 1970-2005/May W5
         (c) 2005 Elsevier Eng.  Info. Inc.
File  35:Dissertation Abs Online 1861-2005/May
         (c) 2005 ProQuest Info&Learning
File  65:Inside Conferences 1993-2005/Jun W1
         (c) 2005 BLDSC all rts. reserv.
File   2:INSPEC 1969-2005/May W5
         (c) 2005 Institution of Electrical Engineers
File  94:JICST-EPlus 1985-2005/Apr W3
         (c)2005 Japan Science and Tech Corp(JST)
File   6:NTIS 1964-2005/May W5
         (c) 2005 NTIS, Intl Cpyrght All Rights Res
File 144:Pascal 1973-2005/May W4
         (c) 2005 INIST/CNRS
File 434:SciSearch(R) Cited Ref Sci 1974-1989/Dec
         (c) 1998 Inst for Sci Info
File  34:SciSearch(R) Cited Ref Sci 1990-2005/May W5
         (c) 2005 Inst for Sci Info
File  99:Wilson Appl. Sci & Tech Abs 1983-2005/May
         (c) 2005 The HW Wilson Co.
File 266:FEDRIP 2005/Jun
         Comp & dist by NTIS, Intl Copyright All Rights Res
File  95:TEME-Technology & Management 1989-2005/Apr W4
         (c) 2005 FIZ TECHNIK
File 438:Library Lit. & Info. Science 1984-2005/May
         (c) 2005 The HW Wilson Co

Set     Items   Description
S1     629174   DIGRAPH? ? OR (DI OR DIRECTED)(1W)GRAPH? ? OR TREE? ? OR B-
              ST? ? OR HIERARCH???
S2       4016   S1(5N)(WALK??? OR TRAVERS??? OR NAVIGAT??? OR BROWS???)
S3      46170   (GRAPHIC?? OR VISUALIZ? OR VISUALIS? OR DISPLAY??? OR VIEW-
              ???)(7N)(NODE? ? OR OBJECT? ? OR S1)
S4       1217   (PARENT? ? OR ROOT? ?)(1W)(NODE? ? OR OBJECT? ?)
S5        549   (CHILD? ? OR CHILDREN OR SUBORDINATE? ?)(1W)(NODE? ? OR OB-
              JECT? ?)
S6      31666   (START??? OR BEGIN? ? OR BEGINNING OR INPUT OR SELECT??? OR
               PICK??? OR CHOOS??? OR CHOSEN OR FOCUS)(5N)(NODE? ? OR OBJEC-
              T? ?)
S7       9915   (1ST OR FIRST OR ONE)(5W)(HIERARCHY OR HIERARCHIES OR TREE?
               ? OR SUBTREE? ?)
S8       1218   LEFT????(5N)(HIERARCHY OR HIERARCHIES OR TREE? ? OR SUBTRE-
              E? ?)
S9      11299   (2ND OR SECOND? OR ANOTHER OR DIFFERENT OR SEPARATE OR ADJ-
              ACENT)(5W)(HIERARCHY OR HIERARCHIES OR TREE? ? OR SUBTREE? ?)
S10      1311   RIGHT????(5N)(HIERARCHY OR HIERARCHIES OR TREE? ? OR SUBTR-
              EE? ?)
S11     15247   (SHARE? ? OR COMMON)(5N)(NODE? ? OR OBJECT? ? OR FOLDER? ?)
S12         1   S2:S3 AND S4 AND S5 AND S6
S13         0   S2:S3 AND PARENT AND CHILD??? AND S6
S14     39353   (START??? OR BEGIN? ? OR BEGINNING OR INPUT OR SELECT??? OR
               PICK??? OR CHOOS??? OR CHOSEN OR FOCUS)(7N)(NODE? ? OR OBJEC-
              T? ?)
S15         4   S2:S3 AND PARENT AND CHILD??? AND S14
S16         2   RD (unique items)
S17         1   S7:S8(10N)S11(10N)S9:S10
```

16/5/1 (Item 1 from file: 8)
DIALOG(R)File 8:Ei Compendex(R)

05233861 E.I. No: EIP99020015434
Title: Recursive representation and progressive display of binary objects for efficient network browsing
Author: Chen, I-Pin; Chen, Zen
Corporate Source: Natl Chiao Tung Univ, Hsinchu, Taiwan
Source: Journal of Visual Communication and Image Representation v 9 n 4 Dec 1998. p 271-286
Publication Year: 1998
CODEN: JVCRE7 ISSN: 1047-3203
Language: English
Document Type: JA; (Journal Article) Treatment: T; (Theoretical)
Journal Announcement: 9904W3
Abstract: When binary objects are browsed in a network environment, data transmission rate, progressive display capability, and view modification under rotation, scaling, and/or translation (R/S/T) are the major factors for **selection** of an appropriate representation model of binary **objects** . A new half-plane-based representation and **display** method for 2D binary **objects** is proposed. Within this modeling framework, a binary object approximated by a shape of a polygon can be represented as a collection of half-planes defined over the edges of the polygon under operations of union and intersection. The basic shape attributes of the object model are the slope and the y-intercept of the boundary line of the constituent half planes. In the progressive **display** of the binary **object** our method adopts the quadtree block subdivision to divide the object image into hierarchical levels of detail (or resolution). Our method determines the color of a quadtree node based on the (angle, intercept) representation parameters. It is shown that the representation parameters at the **parent** node are recursively related to those at the **child** nodes. This recursive relation is crucial for deriving the color of the **nodes** for progressive **object display** . Lemmas for the **node** color determination for an object expressed in the form of half-planes, a convex polygon, or a concave polygon are derived step by step. Our method is generally better than many existing methods in terms of data transmission rate, progressive display capability, and view modification under R/S/T variations. Simulation results are provided to illustrate the performance of our method. (Author abstract) 20 Refs.
Descriptors: *Image communication systems; Object recognition; Data structures; Data reduction; Computational geometry; Approximation theory; Mathematical models; Computer simulation; Web browsers; Color image processing
Identifiers: Two dimensional binary objects
Classification Codes:
741.1 (Light/Optics); 723.5 (Computer Applications); 723.2 (Data Processing); 921.4 (Combinatorial Mathematics, Includes Graph Theory, Set Theory); 921.6 (Numerical Methods)
741 (Optics & Optical Devices); 723 (Computer Software); 921 (Applied Mathematics)
74 (OPTICAL TECHNOLOGY); 72 (COMPUTERS & DATA PROCESSING); 92 (ENGINEERING MATHEMATICS)

01586714 ORDER NO: AADNN-18919

COMPUTABILITY AND COMPLEXITY RESULTS FOR AGREEMENT PROBLEMS IN SHARED MEMORY DISTRIBUTED SYSTEMS (CONSENSUS HIERARCHY, MEMORY ARCHITECTURE)

Author: SCHENK, ERIC
Degree: PH.D.
Year: 1996
Corporate Source/Institution: UNIVERSITY OF TORONTO (CANADA) (0779)
Adviser: FAITH FICH
Source: VOLUME 58/06-B OF DISSERTATION ABSTRACTS INTERNATIONAL.
PAGE 3153. 101 PAGES
Descriptors: COMPUTER SCIENCE
Descriptor Codes: 0984
ISBN: 0-612-18919-8

Agreement problems are central to the study of wait-free protocols for shared memory distributed systems. We examine two specific issues arising out of this study.

We consider the complexity of the wait-free approximate agreement problem in an asynchronous shared memory comprised of only single-bit multi-writer multi-reader registers. For real-valued inputs of magnitude at most s and a real-valued accuracy requirement $\varepsilon>0$ we show matching upper and lower bounds of $\Theta(\log(s/\varepsilon))$ steps and shared registers. For inputs drawn from any fixed finite range this is significantly better than the best possible algorithm for single-writer multi-reader registers, which, for n processes, requires $\Omega(\log\ n)$ steps. These results are used to show a separation between the wait-free single-writer multi-reader and wait-free multi-writer multi-reader models of computation.

The consensus hierarchy characterizes the strength of a **shared object** by its ability to solve the consensus problem in a wait-free manner. **One** important application of a **hierarchy** classifying the power of objects is to compare the power of systems offering **different** collections of objects. Ideally, a **hierarchy** should reduce the task of determining the strength of an architecture supporting shared memory distributed systems to the problem of determining the strength of each type of shared object supported by the architecture. Informally, a hierarchy that allows this is robust. Several variations of the consensus hierarchy have appeared in the literature, and it has been shown that all but one of them are not robust. The remaining hierarchy, named $h\sbsp{m}{r},$ has been the subject of considerable research. We show that, in a natural setting, the consensus hierarchy $h\sbsp{m}{r}$ is not robust.

```
File 275:Gale Group Computer DB(TM) 1983-2005/Jun 03
         (c) 2005 The Gale Group
File 621:Gale Group New Prod.Annou.(R) 1985-2005/Jun 03
         (c) 2005 The Gale Group
File 636:Gale Group Newsletter DB(TM) 1987-2005/Jun 03
         (c) 2005 The Gale Group
File  16:Gale Group PROMT(R) 1990-2005/Jun 03
         (c) 2005 The Gale Group
File 160:Gale Group PROMT(R) 1972-1989
         (c) 1999 The Gale Group
File 148:Gale Group Trade & Industry DB 1976-2005/Jun 03
         (c)2005 The Gale Group
File 624:McGraw-Hill Publications 1985-2005/Jun 06
         (c) 2005 McGraw-Hill Co. Inc
File  15:ABI/Inform(R) 1971-2005/Jun 06
         (c) 2005 ProQuest Info&Learning
File 647:CMP  Computer Fulltext 1988-2005/May W4
         (c) 2005 CMP Media, LLC
File 674:Computer News Fulltext 1989-2005/May W5
         (c) 2005 IDG Communications
File 696:DIALOG Telecom. Newsletters 1995-2005/Jun 04
         (c) 2005 The Dialog Corp.
File 369:New Scientist 1994-2005/Apr W2
         (c) 2005 Reed Business Information Ltd.

Set     Items   Description
S1     442056   DIGRAPH? ? OR (DI OR DIRECTED)(1W)GRAPH? ? OR TREE? ? OR B-
              ST? ? OR HIERARCH???
S2       5357   S1(5N)(WALK??? OR TRAVERS??? OR NAVIGAT??? OR BROWS???)
S3      59425   (GRAPHIC?? OR VISUALIZ? OR VISUALIS? OR DISPLAY??? OR VIEW-
              ???)(7N)(NODE? ? OR OBJECT? ? OR S1)
S4        970   (PARENT? ? OR ROOT? ?)(1W)(NODE? ? OR OBJECT? ?)
S5        486   (CHILD? ? OR CHILDREN OR SUBORDINATE? ?)(1W)(NODE? ? OR OB-
              JECT? ?)
S6      32862   (START??? OR BEGIN? ? OR BEGINNING OR INPUT OR SELECT??? OR
              PICK??? OR CHOOS??? OR CHOSEN OR FOCUS)(5N)(NODE? ? OR OBJEC-
              T? ?)
S7       9706   (1ST OR FIRST OR ONE)(5W)(HIERARCHY OR HIERARCHIES OR TREE?
              ? OR SUBTREE? ?)
S8       2322   LEFT????(5N)(HIERARCHY OR HIERARCHIES OR TREE? ? OR SUBTRE-
              E? ?)
S9       6306   (2ND OR SECOND? OR ANOTHER OR DIFFERENT OR SEPARATE OR ADJ-
              ACENT)(5W)(HIERARCHY OR HIERARCHIES OR TREE? ? OR SUBTREE? ?)
S10      3023   RIGHT????(5N)(HIERARCHY OR HIERARCHIES OR TREE? ? OR SUBTR-
              EE? ?)
S11     29607   (SHARE? ? OR COMMON)(5N)(NODE? ? OR OBJECT? ? OR FOLDER? ?)
S12        13   S2:S3(50N)S4(50N)S5(50N)S6
S13     41074   PARENT(20N)CHILD???
S14        31   S2:S3(50N)S13(50N)S6
S15        35   S12 OR S14
S16        27   RD (unique items)
S17        26   S16 NOT PY=2003:2005
S18         2   S7:S8(10N)S11(10N)S9:S10
S19         1   RD (unique items)
```

17/3,K/1 (Item 1 from file: 275)
DIALOG(R)File 275:Gale Group Computer DB(TM)

02106601 SUPPLIER NUMBER: 19774290 (USE FORMAT 7 OR 9 FOR FULL TEXT)
Surface modeling. (includes related article on basic surface types) (Technology Information)
Rowe, Jeffrey
Computer Graphics World, v20, n9, p47(5)
Sep, 1997
ISSN: 0271-4159 LANGUAGE: English RECORD TYPE: Fulltext; Abstract
WORD COUNT: 2593 LINE COUNT: 00218

... add holes, protrusions, blends, or other characteristics. on the stitched solid, these too would be regenerated. Ideally, **parent / child** relationships will be **displayed** graphically and as feature- **tree** dependencies.
Associativity also brings up the topic of unification -- the marriage of surfaces and parametric solids to...

...says Autodesk's Klemm. "Each object is isolated and there is no hierarchial tree (defining) how the **object** was built. We are **starting** to move toward this 'feature-based' approach by defining a set of objects that may be modified...

17/3,K/2 (Item 2 from file: 275)
DIALOG(R)File 275:Gale Group Computer DB(TM)

01995079 SUPPLIER NUMBER: 18791180 (USE FORMAT 7 OR 9 FOR FULL TEXT)
3D Studio MAX. (Kinetix animation package)(one of eight evaluations of 3D tools in "Ideas Taking Shape 3D Animation Software") (Software Review)(Evaluation)
Grunin, Lori
PC Magazine, v15, n19, p220(2)
Nov 5, 1996
DOCUMENT TYPE: Evaluation ISSN: 0888-8507 LANGUAGE: English
RECORD TYPE: Fulltext; Abstract
WORD COUNT: 989 LINE COUNT: 00081

... was fairly easy, too. To create an inverse kinematics chain or a standard hierarchical chain, you simply **select** the **parent object** , click on the link icon, and drag a connection between that and the **child object** . You set inheritance and locking parameters for rotation, translation, and scaling on an axis-by-axis basis...

...in which you can drag and drop chains from one parent to another, 3D Studio MAX's **Hierarchy view** forces you to break and re-create chains (unless you simply want to reverse the order of...

17/3,K/3 (Item 3 from file: 275)
DIALOG(R)File 275:Gale Group Computer DB(TM)

01966973 SUPPLIER NUMBER: 18564806
Components, we got components.(Desktop DBMS)(Sheridan Software Systems Inc's ClassAssist, sp Assist and VBAssist components for Visual Basic development)(Column)
Spitzer, Tom
DBMS, v9, n9, p107(4)
August, 1996
DOCUMENT TYPE: Column ISSN: 1041-5173 LANGUAGE: English
RECORD TYPE: Fulltext; Abstract

WORD COUNT: 3912 LINE COUNT: 00311

... Index") into which it encodes the **parent - child** hierarchy. Each record represents a node in the outline, and has a **parent** ID, a level ID, and a sequence number to assign its order within the level. In addition...

...various data elements being displayed, the contents of the rightmouse menu that should be invoked when various **nodes** are **selected** , and style information to apply to the **nodes** . When VB **displays** a form containing the InfoSleuth control, it builds the outline from the information in the Sleuth...

17/3,K/4 (Item 4 from file: 275)
DIALOG(R)File 275:Gale Group Computer DB(TM)
(c) 2005 The Gale Group..All rts. reserv.

01902057 SUPPLIER NUMBER: 17945903 (USE FORMAT 7 OR 9 FOR FULL TEXT)
Use a decision tree class for complex logic. (Technology Tutorial)
Hill, Robert
Data Based Advisor, v14, n3, p64(4)
March, 1996
ISSN: 0740-5200 LANGUAGE: English RECORD TYPE: Fulltext; Abstract
WORD COUNT: 1922 LINE COUNT: 00162

... ID, a result string that is the return value of the traverse method, and pointers to its **parent** , its **children** , and its direct sibling. It also contains a pointer to a function that takes a string as... method to evaluate a function that returns an integer value. Here's the Delphi version of the **traverse** () method:

```
    { Traverse (Execute) the Decision Tree function TDecisionTree:
traverse : string; var
    response, lastResult: integer;
    curNode, parent : PTreeNode;
    Func: TLogFunction; begin
    self.curNode :. self.rootNode;
     parent :. self.rootNode;
    lastresult : = 0;
    while self.curNode
    . Child <> nil do begin
    { Evaluate the Function
    Func := self.curNode
    .logical;
    if @Func <> nil then
    response : = Func(self.curNode
    .ID, lastResult)
    else
    response : = 0;
    if response = 0 then
    { Return to the Parent Node }
    self.curNode := self.curNode
    . Parent
    else
    { Move to the specified child }
    self.moveToChild(response);
    lastResult : = response;
    end;
    { Assign the return value }
    if self.curNode
    .Result <> `` then
    traverse : = self.curNode
    .Result
    else
    traverse : = ` No Solution'; end;
```

The traverse() method **starts** at the root **node** of the decision tree

and evaluates the function at each node it encounters. The node's function ...

17/3,K/5 (Item 5 from file: 275)
DIALOG(R)File 275:Gale Group Computer DB(TM)

01797690 SUPPLIER NUMBER: 17016779 (USE FORMAT 7 OR 9 FOR FULL TEXT)
Menu madness, or the mystery of MDI. (Multiple Document Interface)(programming in CA-Visual Objects)(Tutorial)
Spence, Rick
Data Based Advisor, v13, n4, p128(4)
May, 1995
DOCUMENT TYPE: Tutorial ISSN: 0740-5200 LANGUAGE: ENGLISH
RECORD TYPE: FULLTEXT; ABSTRACT
WORD COUNT: 2313 LINE COUNT: 00176

... types of windows, a ShellWindow called StandardShellWindow, and a data window called StdDataWindow.
StandardShellWindow is the MDI **parent** window; that is, it's the window on whose canvas the **child** windows are displayed. StdDataWindow is the MDI **child** window. StdDataWindow's owner is StandardShellWindow.
You can attach menus to both MDI **child** windows and **parent** windows. Each **child** window can have a different menu, and what surprises most programmers is that the **child** window's menu is displayed on the shell window. As you switch **focus** between child windows, Visual **Objects** automatically **displays** the child window's menu on the shell window. When there are no child windows open, Visual **Objects** **displays** the shell window's own menu. A key question you'll face when designing a program's...

17/3,K/6 (Item 6 from file: 275)
DIALOG(R)File 275:Gale Group Computer DB(TM)

01712233 SUPPLIER NUMBER: 16250411 (USE FORMAT 7 OR 9 FOR FULL TEXT)
Trio meets diverse demands; gains and glitches characterize IBM's Warp 3, Microsoft's NT 3.5, Apple's System 7.5. (desktop operating systems) (Software Review) (includes related article on test methodology) (Evaluation)
Coffee, Peter
PC Week, v11, n45, p138(6)
Nov 14, 1994
DOCUMENT TYPE: Evaluation ISSN: 0740-1604 LANGUAGE: ENGLISH
RECORD TYPE: FULLTEXT; ABSTRACT
WORD COUNT: 4512 LINE COUNT: 00357

... Program Manager, by contrast, begins with the tool and uses the tool to find and open the **object** .
Warp let us **choose** to make a **parent** window close whenever a **child** window opened, making displays seem less crowded. New users might also be comforted by Warp's new...

...other modules. Unlike ClarisWorks, IBM Works is tool-centric: Dragging spreadsheet cells into a document yielded a **graphical** **object** , rather than a table, letting fonts and column widths be controlled only from the spreadsheet (and then...

17/3,K/7 (Item 7 from file: 275)
DIALOG(R)File 275:Gale Group Computer DB(TM)

01614842 SUPPLIER NUMBER: 14203403 (USE FORMAT 7 OR 9 FOR FULL TEXT)

Inside SQLWindows 4.0: SQLWindows 4.0 sends Gupta Corporation to the head of the class in Microsoft Windows client/server development. (Software Review) (Evaluation)
McClanahan, David
DBMS, v6, n10, p54(7)
Sept, 1993
DOCUMENT TYPE: Evaluation ISSN: 1041-5173 LANGUAGE: ENGLISH
RECORD TYPE: FULLTEXT; ABSTRACT
WORD COUNT: 6485 LINE COUNT: 00523

... serves as the starting window for the design of a new application. This represents the user's **view** of the application; you can add window **objects** to this form and create other forms for the application. The objects you can place on a...

...supports MDI (Multiple Document Interface) windows, which allows you to add toolbars and palettes. MDI allows a **parent** "frame" window to contain **child** windows, each of which you can move, resize, tile, minimize, or maximize.

The SQLWindows Tool Palette **displays** icons that represent the tools for placing **objects** in the form window. To add an object, you click on the corresponding icon (such as a...

...the form to insert and size the object. To modify an object, use the Tool Palette's **Object Selector** and click on the **object** ; to resize or move an **object** , **select** Window Grabber; and to duplicate an **object** (including its code), **select** **Object** Duplicator, click on the **object** , and then drag and release at the new position. Clicking the Form push button in the Tool...

17/3,K/8 (Item 8 from file: 275)
DIALOG(R)File 275:Gale Group Computer DB(TM)
(c) 2005 The Gale Group. All rts. reserv.

01600733 SUPPLIER NUMBER: 13741270 (USE FORMAT 7 OR 9 FOR FULL TEXT)
X-ray vision for networks. (VisiSoft's VisiNet 2.0 network management software)(includes executive summary) (Software Review) (Test Drive) (Evaluation)
Henderson, Tom
LAN Magazine, v8, n6, p209(4)
June, 1993
DOCUMENT TYPE: Evaluation ISSN: 0898-0012 LANGUAGE: ENGLISH
RECORD TYPE: FULLTEXT; ABSTRACT
WORD COUNT: 2617 LINE COUNT: 00208

... bring VisiNet to life, I turned to the tutorial, where I found that VisiNet has three basic **objects** : Views, Nodes, and Links. You create a View, populate the View with Nodes, then link the objects...

...looped View by mistake, the documentation warns you at every possible turn.

You can populate Views with **objects** **selected** from the standard ones that come with one of the default .NET files, or you can define them yourself. You can easily move member **objects** from one **view** to another by using drag-and-drop techniques.

VisiNet suggests you establish a hierarchical relationship among the Views. For example, a Parent **View** might consist of "System Information" **objects** , whose Child **Views** have information related to the Parent's genre. One of LAN Magazine's criticisms of VisiNet 1...

...true. The latest version has stored x-ray vision; it now remembers its children.

I created a **View** called Master that contained six **objects** : one for users and groups, one for file servers, one for print servers and print

queues, another...

...workstation connections, one containing the LANMAG network users, and one for the Beach Labs network.
For each **object** in the Master **View** , I created an associated Child **View** . All **Views** can be automatically populated with **nodes** by **choosing** Logging from the main options set, then Discovery, an option whereby VisiNet explores network components and databases...

17/3,K/9 (Item 9 from file: 275)
DIALOG(R)File 275:Gale Group Computer DB(TM)
(c) 2005 The Gale Group. All rts. reserv.

01512418 SUPPLIER NUMBER: 12226657 (USE FORMAT 7 OR 9 FOR FULL TEXT)
Through the looking glass: VisiSoft's VisiNet lets you take your network's pulse and blood pressure. (network management software) (Software Review) (Test Drive) (Evaluation)
Mackin, Ken
LAN Magazine, v7, n4, p140(4)
April, 1992
DOCUMENT TYPE: Evaluation ISSN: 0898-0012 LANGUAGE: ENGLISH
RECORD TYPE: FULLTEXT; ABSTRACT
WORD COUNT: 2866 LINE COUNT: 00222

... and imported via Clipboard. I created a custom map for LAN'S San Francisco office as a **child** view with the U.S. view as a **parent** .
Background maps must be imported into the specific VisiNet file you are using before they can be...

...times before I had all the maps needed for the test drive. How about a multiple-item **pick** list?
SWOLLEN WITH **NODES**
Once your **view** is established, fill it with **nodes** . A **node** is an object that refers to anything on the LAN, from users, to groups, to workstations, with...

17/3,K/10 (Item 10 from file: 275)
DIALOG(R)File 275:Gale Group Computer DB(TM)
(c) 2005 The Gale Group. All rts. reserv.

01504179 SUPPLIER NUMBER: 11973768 (USE FORMAT 7 OR 9 FOR FULL TEXT)
MasterMind: improving the search. (Tutorial)
Montgomery, George
AI Expert, v7, n4, p40(8)
April, 1992
DOCUMENT TYPE: Tutorial ISSN: 0888-3785 LANGUAGE: ENGLISH
RECORD TYPE: FULLTEXT; ABSTRACT
WORD COUNT: 3387 LINE COUNT: 00337

... steps to the methodology are as follows:
1. Organize the state space in tree hierarchy in which **child** **nodes** represent refinements to the more general state of the **parent** **node** , and then **choose** a knowledge representation scheme that fits the state space description.
2. Investigate the size of the state...

...which the same problem is represented in a smaller state space.
3. Define operators that transform a **parent** **node** into its **child** **nodes** and allow **traversal** of the **tree** structure.
4. Define predicates that can test branch nodes (as well as fringe nodes) in the tree...

17/3,K/11 (Item 11 from file: 275)
DIALOG(R)File 275:Gale Group Computer DB(TM)

01421308 SUPPLIER NUMBER: 09767465 (USE FORMAT 7 OR 9 FOR FULL TEXT)
The first real animation alternative. (Paracomp Inc.'s FilmMaker animation software) (Software Review) (evaluation)
Murie, Michael
MacWEEK, v5, n1, p104(3)
Jan 8, 1991
DOCUMENT TYPE: evaluation ISSN: 0892-8118 LANGUAGE: ENGLISH
RECORD TYPE: FULLTEXT; ABSTRACT
WORD COUNT: 1366 LINE COUNT: 00109

... are not traditional.
Animate and Mark. The Animate program is the central FilmMaker application. The movement of **graphical objects** is defined in this module. But Animate does not recognize standard Macintosh graphics formats; they must first...

...size of the graphic is displayed. If you need to see the position of parts of a **graphical object** while in Animate, the Mark application provides a line-drawing tool for "marking" a stick-figure-style outline of the graphic. This outline will then be visible in Animate.
Once inside Animate, **objects** are created that represent each of the **graphical** items in an animation. The parent **object** to which everything is attached is the stage; **child objects** are attached to the **parent**, and **objects** can in turn be attached to the children for a hierarchical series of dependencies. **Child objects** are dependent upon their parent. They can be on the stage only during the lifetime of the...

...is also applied to the children. This makes it possible to create very complicated movements.
Once an **object** has been created, it is assigned a **graphics** file. This may be a single graphic or a series of frames already created by Animate. While...

...tools. Objects are moved, resized and rotated using the animation tools. These tools can be applied by **selecting** the **object** with the tool and dragging, but you have much more control if you double-click on the...

17/3,K/12 (Item 12 from file: 275)
DIALOG(R)File 275:Gale Group Computer DB(TM)

01385119 SUPPLIER NUMBER: 08785076 (USE FORMAT 7 OR 9 FOR FULL TEXT)
vsDesigner Expert. (Sage Software Inc.'s computer-aided software engineering tool) (Software Review) (evaluation)
Lewis, Scott
Computer Language, v7, n9, p95(7)
Sept, 1990
DOCUMENT TYPE: evaluation ISSN: 0749-2839 LANGUAGE: ENGLISH
RECORD TYPE: FULLTEXT; ABSTRACT
WORD COUNT: 3882 LINE COUNT: 00309

... prove sufficient for most designs.
To build a design, first place the objects in the top-level **node** by selecting them from a symbols menu. vsDesigner displays prompts with information on how to place each corresponds to parent-object data flow. Objects in **child nodes** can have children of their own as a design is decomposed into more detail. A **child node** can effectively have several parents, eliminating the need for duplicate subordinate branches in a hierarchy. If the...

...is well implemented. To travel down the hierarchy, simply select Roam, hit F3, and then select Downtree. **Objects** with **child** nodes are **displayed** with dotted outlines. Move the cursor to an **object** and click twice with the mouse. vsDesigner **displays** the **child** node , again with dotted outlines for objects with children. To move back up, simply select Uptree.

Each node...

...source of each syntax, a reference list, brief explanations of applicable attributes, and a drawing of each **object** as it would be **displayed** by vsObject Maker, along with a one- or two-line description and a list of attributes. And...

17/3,K/13 (Item 13 from file: 275)
DIALOG(R)File 275:Gale Group Computer DB(TM)

01380529 SUPPLIER NUMBER: 09582897 (USE FORMAT 7 OR 9 FOR FULL TEXT)
An overview of the HP interactive visual interface.
Lau, Roger K.; Thompson, Mark E.
Hewlett-Packard Journal, v41, n5, p6(3)
Oct, 1990
ISSN: 0018-1153 LANGUAGE: ENGLISH RECORD TYPE: FULLTEXT; ABSTRACT
WORD COUNT: 1755 LINE COUNT: 00142

... of primitive objects. This includes menus and their component menu panes, row-columns, and scroll lists.

* Primitive **Objects** . These are basic widgets and **graphics objects** --the basic visual pieces that make up the display. Graphics primitives include items such as polylines, splinesand scrollbars. Both types of primitive **objects** can receive **input** from the user.

* Low-Level **Objects** . These are mostly nonvisual objects that are used to specify certain object attributes. Objects that handle object data structures and objects that handle events are examples of low-level **objects** .

Because an **object hierarchy** is used, **displays** can be created from the top down (**parent** to **child**) or the bottom up (**child** to **parent**), giving the designer a lot of flexibility in implementation. Certain objects can be gathered and arranged by making them into **children** of composite objects. Composite objects can be used to organize and add extra control over their descendant...

17/3,K/14 (Item 14 from file: 275)
DIALOG(R)File 275:Gale Group Computer DB(TM)

01376759 SUPPLIER NUMBER: 09485339 (USE FORMAT 7 OR 9 FOR FULL TEXT)
QuarkXPress 3.0: in its third version, QuarkXPress moves to the top of the heap in page layout. (Quark's desktop publishing program) (Software Review) (evaluation)
Taub, Eric
MacUser, v6, n11, p52(2)
Nov, 1990
DOCUMENT TYPE: evaluation ISSN: 0884-0997 LANGUAGE: ENGLISH
RECORD TYPE: FULLTEXT; ABSTRACT
WORD COUNT: 1488 LINE COUNT: 00113

... can even create objects that stretch across multipage spreads.

You're no longer confined to using cumbersom **parent / child** boxes when you want to associate **objects** permanently. You can now **select** and permanently group multiple **objects** with the standard Shift-click command or by using the marquee tool. Although the **parent / child** box arrangement is still available, users can opt to use the more flexible - and less

confusing - option of permanently grouping objects as necessary.
QuarkXPress now lets you rotate any **object** or group either text or **graphic**) in increments as fine as .001 degree, either by using the mouse with a new rotation tool...

17/3,K/15 (Item 15 from file: 275)
DIALOG(R)File 275:Gale Group Computer DB(TM)

01346385 SUPPLIER NUMBER: 08093350 (USE FORMAT 7 OR 9 FOR FULL TEXT)
Super 3D and Swivel 3D: these excellent 3-D tools complement each other perfectly. Serious 3-D artists should get both. (Software Review) (three-dimensional; graphics packages from Silicon Beach Software and Paracomp, respectively) (evaluation)
Parascandolo, Salvatore
MacUser, v6, n3, p65(2)
March, 1990
DOCUMENT TYPE: evaluation ISSN: 0884-0997 LANGUAGE: ENGLISH
RECORD TYPE: FULLTEXT; ABSTRACT
WORD COUNT: 1302 LINE COUNT: 00099

... objects. Grids for both attitude and position help objects snap into place without tweaking.
All Swivel 3D **objects** are constructed in a special **view** , with the main scene out of sight. You **start** defining an **object** by drawing its cross section. A bat, a ball, or a coin, for example, all have circular...

...outline views at any time. It's a powerful shaping system that sounds simple, but what an **object** 's three **views** should look like isn't always obvious -- and you'll definitely have to experiment.
Swivel 3D lets you link items in several ways. Every link has a " **parent** " **object** and a " **child** " **object** . Links can be stiff, as in conventional groups; ball-jointed, in which the components pivot but don...

...rubbery, in which the child elements move and turn independently but aren't left behind when the **parent object** moves. You can even specify how much freedom each **child object** is allowed relative to its parent -- a door might swing no more than 90 degrees, for example...

17/3,K/16 (Item 16 from file: 275)
DIALOG(R)File 275:Gale Group Computer DB(TM)

01315943 SUPPLIER NUMBER: 07577946 (USE FORMAT 7 OR 9 FOR FULL TEXT)
The NewWave Object Management Facility.
Dysart, John A.
Hewlett-Packard Journal, v40, n4, p17(7)
August, 1989
ISSN: 0018-1153 LANGUAGE: ENGLISH RECORD TYPE: FULLTEXT; ABSTRACT
WORD COUNT: 4382 LINE COUNT: 00332

... The object tells the OMF that it is closing, and then removes its user interface from the **display** screen.
* Termination. An object remains active as long as it is open or is being held active...to activate and send messages to a vierw than to activate and send messages to the child **object** of the **view** . The messages may ultimately be routed to the child, but they may instead be routed to a...

...snapshot is an object that serves as an intelligent buffer that allows the linked data from a **child object** to be accessed without activating the child. This results in better performance and use of resources. When a **view** is initialized, the **child object** tells the OMF whether or not to

create a snapshot for the view, and if so, what kind of snapshot is desired. The **child object** then sends messages containing the linked data information to the snapshot. When the parent **object** sends a **view** message to the **view** , the OMF routes the message to the snapshot. The **child object** remains inactive. Fig. 8 illustrates how a snapshot is associated with a view.

One reason why snapshots...

17/3,K/17 (Item 17 from file: 275)
DIALOG(R)File 275:Gale Group Computer DB(TM)

01315942 SUPPLIER NUMBER: 07577944 (USE FORMAT 7 OR 9 FOR FULL TEXT)
An object-based user interface for the HP NewWave environment.
(Hewlett-Packard Co.)
Showman, Peter S.
Hewlett-Packard Journal, v40, n4, p9(9)
August, 1989
ISSN: 0018-1153 LANGUAGE: ENGLISH RECORD TYPE: FULLTEXT; ABSTRACT
WORD COUNT: 4748 LINE COUNT: 00374

... the OMF.

Although there are a number of ways links could be managed, a hierarchical structure oflinked **objects** is used as the **starting** point for the NewWave environment. The NewWave Office, which is a special application that provides access to...

...of linked objects, the one that is closer to the top of the hierarchy is called the **parent** and the lower one is the **child** . In general, the set of all the objects below a given object might be called its descendants...

...keep the objects together: these are called simple links. Links can also be used to let the **child** object provide data or services to the **parent** object; these are called data links. Because a data link in effect allows the **parent** to **view** a portion of the **child object** , data links are also sometimes called **views** . These provide for automatic updating of one **object** by another, a facility sometimes referred to as a hot connect.

The Office Metaphor

Rather than requiring users to manipulate links directly, which would have required us to **display hierarchy** diagrams such as that shown in Fig. 2, the NewWave Office provides an office metaphor for managing...

...no information passes between these objects, they can be connected by simple links. And because the folder **displays** no data from the contained objects, an object within a folder can be represented simply as an...

...the NewWave Office (See Fig. 4).

From any level, the user can manipulate a lower-level child **object** as a whole entity by manipulating the icon. Items can be moved from one folder to another...by the user. All the user needs to worry about is where the icons that represent the **objects** are **displayed** .

The user can also choose to open a contained object to operate on it in detail, by...

...and provides the name of the data file. The application program then creates a new window and **displays** the contents of the open **object** for the user to manipulate as desired (see Fig. 5). The representation in the parent object is...

...and tables contained by the document are all attached to it by a type of data link (**view**) called a visual link. These contained **child objects** are in fact responsible for **displaying** and printing the information contained in the corresponding illustrations and tables. Therefore, as in the first example, because the **child object** is linked to its own

software, the user can double-click on the illustration to open the corresponding **child object** , thus running the associated application, and make changes to the illustration using the application's user interface ...

...of the sample applications provided to NewWave developers, Layout and HP Shape. when the user closes the **child object** , the application associated with the child first updates and representation in the parent document automatically.
Example 3...

17/3,K/18 (Item 1 from file: 621)
DIALOG(R)File 621:Gale Group New Prod.Annou.(R)
(c) 2005 The Gale Group. All rts. reserv.

03276925 Supplier Number: 92426531 (USE FORMAT 7 FOR FULLTEXT)
Tom Sawyer Software Releases Layout Assistant for Visio; Layout Assistant Produces Beautiful Diagrams Within Visio.
PR Newswire, pSFTH10904102002
Oct 4, 2002
Language: English Record Type: Fulltext
Document Type: Newswire; Trade
Word Count: 380

... layout styles solve the most common diagram formatting needs
-- Circular layout displays clusters in networks
-- Hierarchical layout **displays** dependencies and flows
-- **Tree** layout **displays** **parent** and **child** relationships
-- Layout properties enable adjustment of diagram formatting
The Layout Assistant, Professional Edition, expands upon the Standard...

...structured Hierarchical and Orthogonal layout
-- Enhanced connection points permit exact attachment points with Hierarchical and Orthogonal layout
-- **Selected object** layout properties enable you to adjust diagram appearance at the object level
Brendan Madden, Chief Executive Officer...

17/3,K/19 (Item 1 from file: 636)
DIALOG(R)File 636:Gale Group Newsletter DB(TM)
(c) 2005 The Gale Group. All rts. reserv.

01414809 Supplier Number: 41845291 (USE FORMAT 7 FOR FULLTEXT)
NYNEX: LAUNCHES ENHANCED VERSION OF INTEGRATED NETWORK MANGEMENT SYSTEM; NYNEX ALLINK OPERATIONS COORDINATOR 2.0
EDGE, on & about AT&T, v6, n132, pN/A
Feb 4, 1991
Language: English Record Type: Fulltext
Document Type: Newsletter; Trade
Word Count: 422

... New Features and Enhancements: Generic rule types for analyzing events to report alarms
- Correlation of events against **children** network **objects** to report an alarm against the **parent object** .
- Correlation of the inaccessibility of downstream network objects with an event reported against an upstream object.
- Correlation...

...particular network object with the other network objects to which it is

connected.
* Suppression of events from **selected** types of network **objects** based on time-of-day or operator selection.
* A telescoping **graphical view tree** of the network, showing the network **objects** and their association with users of the network.
* Textual display of the network users associated with a network **object selected** by either mouse **selection** from a **graphical** view, by keyboard entry of the element name, or by mouse selection from the textual alarm summary...

17/3,K/20 (Item 1 from file: 16)
DIALOG(R)File 16:Gale Group PROMT(R)

09754451 Supplier Number: 85410444 (USE FORMAT 7 FOR FULLTEXT)
The Hype over Hyperbolic Browsers.(Brief Article)
Allen, Maryellen Mott
Online, v26, n3, p20(6)
May, 2002
Language: English Record Type: Fulltext
Article Type: Brief Article
Document Type: Magazine/Journal; Professional Trade
Word Count: 3028

... as the Poincare disk model. In a paper published by John Lamping, Ramana Rao, and Peter Pirolli (" **Visualizing** Large **Trees** Using the Hyperbolic **Browser** ," Proceedings of the Conference on Human Factors in Computing Systems, April 13-18, 1996, Vancouver, British Columbia...

...org/sigchi/ chi96/proceedings/video/Lamping/hb-video.html)), the authors explain the mapping of a hierarchical **tree** structure to a hyperbolic **display** citing two significant qualities of the structure:
* The nodes or components of the tree diminish in size the farther away they are from the center of the **display** .
* The number of **nodes** or components grows exponentially from parent to **child** .
The authors further explain, "The hyperbolic **browser** initially **displays** a **tree** with its root in the center, but the **display** can be smoothly transformed to bring other **nodes** into **focus** ... In all cases, the amount of space available to a node falls off as a continuous function ...

17/3,K/21 (Item 2 from file: 16)
DIALOG(R)File 16:Gale Group PROMT(R)

09326692 Supplier Number: 81218235 (USE FORMAT 7 FOR FULLTEXT)
Radiosity, global illumination, and other magic. (3D Visualization).
Boardman, Ted
Cadence, v17, n1, p42(4)
Jan, 2002
Language: English Record Type: Fulltext
Document Type: Magazine/Journal; Trade
Word Count: 2467

... object to another is accomplished by clicking the Link button and left-clicking and holding on the **child object** . While holding the left mouse button, drag the cursor to the **parent** object and release. Special Link cursors and a dotted line will be displayed during the process, as...

...link has occurred, but if you click the Select button and type H, you can check the **Display** Subtree option in the **Select Objects** dialog to **view** the **child** 's name indented below the **parent** name to indicate

hierarchical linking, as shown in Figure 3.

Now when the **parent** object, in this case Sphere01, is moved or rotated, Box01 also moves with or rotates about Sphere01's pivot point. When the **child object**, Box01, is moved or rotated, it is independent of the **parent**.

A good example of hierarchical linking's usefulness is the case of a car with the body...

17/3,K/22 (Item 3 from file: 16)
DIALOG(R)File 16:Gale Group PROMT(R)

05586484 Supplier Number: 48456682 (USE FORMAT 7 FOR FULLTEXT)
YOU CONTROL THE 3D PIPELINE
ABOUAF, JEFFREY
Interactivity, p64
May, 1998
Language: English Record Type: Fulltext
Document Type: Magazine/Journal; Trade
Word Count: 2950

... as FIT ON PICK - a special display mode that lets you center and zoom in on a **selected object**). The righthand section of the toolbar contains floating palettes, including View, Light Source, Color, Infrared, Material, Texture...animation.

Database Management

Every time you add an element of any kind to a scene, a new **node** appears in the **Hierarchy view**. Elements added to a pre-existing **object** or group of objects are represented as children of the **parent object**; that is, their nodes appear below that of their parent. When you create a new 3D shape, **child nodes** representing its faces and vertices appear automatically.

Clicking a **node selects** its counterpart in the **Graphics view** Double-clicking a **node** opens its Attributes dialog for numerical editing. Nodes can be dragged to rearrange their position both vertically...more complex motions, you can use a flipbook animation. This technique captures a sequence of positions as **children** of a **parent** node, then plays them in succession to give the appearance of motion. Flipbook animation is very efficient...

...resolution versions can be swapped in, reducing the workload on the rendering engine. Using the Create LOD **Node** Tool, you **select** a parent **node** in the **Hierarchy view**. This creates an LOD copy that hears a sibling relationship to the original. Then you isolate it...

17/3,K/23 (Item 1 from file: 148)
DIALOG(R)File 148:Gale Group Trade & Industry DB

05108347 SUPPLIER NUMBER: 10381847 (USE FORMAT 7 OR 9 FOR FULL TEXT)
NYNEX: launches enhanced version of integrated network management system; NYNEX ALLINK Operations Coordinator 2.0. (product announcement)
EDGE, on & about AT&T, v6, n132, p13(1)
Feb 4, 1991
DOCUMENT TYPE: product announcement LANGUAGE: ENGLISH
RECORD TYPE: FULLTEXT
WORD COUNT: 221 LINE COUNT: 00039

... New Features and Enhancements: Generic rule types for analyzing events to report alarms
- Correlation of events against **children** network **objects** to report an alarm against the **parent object**.
- Correlation of the inaccessibility of downstream network objects

with an event reported against an upstream object.
- Correlation...

...network
object with the other network objects to which it is connected.
o Suppression of events from **selected** types of network **objects** based on time-of-day or operator selection.
o A telescoping **graphical view tree** of the network, showing the network **objects** and their association with users of the network.
o Textual display of the network users associated with a network object **selected** by either mouse **selection** from a **graphical** view, by
keyboard entry of the element name, or by mouse selection from the textual alarm summary...

17/3,K/24 (Item 1 from file: 15)
DIALOG(R)File 15:ABI/Inform(R)

02325865 86067891
Kanban setting through artificial intelligence: a comparative study of artificial neural networks and decision trees
Markham, Ina S; Mathieu, Richard G; Wray, Barry A
Integrated Manufacturing Systems v11n4 PP: 239-246 2000
ISSN: 0957-6061 JRNL CODE: ING
WORD COUNT: 4908

...TEXT: relatively homogeneous "nodes" by searching for "yes/no" responses to questions involving the predictor variables. Results are **displayed** in a decision **tree** that is useful for communicating the classification decision to others and for automatically classifying or predicting new...

...be generalized as involving the partitioning of data into terminal nodes by a sequence of binary splits, **starting** at a **parent node** . The procedure searches through all values of all the independent variables to obtain the variable and the value that provides the best split into **child nodes** . So, if there are 20 independent variables and 1,000 possible values for each of the variables...

17/3,K/25 (Item 2 from file: 15)
DIALOG(R)File 15:ABI/Inform(R)

00847898 94-97290
Methodology-driven use of automated support in business process re-engineering
Dennis, Alan R; Daniels, Robert M Jr; Hayes, Glenda; Nunamaker, Jay F Jr
Journal of Management Information Systems: JMIS v10n3 PP: 117-138 Winter 1993-1994
ISSN: 0742-1222 JRNL CODE: JMI
WORD COUNT: 8152

...TEXT: the descriptions that exist in the repository for any particular object by moving the cursor over the **object** and **selecting** "description" from a menu. GB also isolates orphan nodes, to help analyze the model for completeness. The...

...a portion of the drawing. Drawings can be linked together to let the user jump from a " **parent** " drawing to " **child** " drawings and back. A single process is exploded into a tree of subprocesses in the **child** drawing. GB is used to **display objects** and relationships already in the model.

CASE STUDY
This section presents the results of using the EA...

17/3,K/26 (Item 3 from file: 15)
DIALOG(R)File 15:ABI/Inform(R)

00607304 92-22407
Recent Developments and Future Directions in Mathematical Programming
Johnson, Ellis L.; Nemhauser, George L.
IBM Systems Journal v31n1 PP: 79-93 1992
ISSN: 0018-8670 JRNL CODE: ISY
WORD COUNT: 9356

...TEXT: fractional variable should be the one on which to make the decision to branch. One way to **view** the subproblem list is as a **tree** in which each subproblem corresponds to a node and in which the immediate descendants of a node...

...natural way to select subproblems is by depth first search plus backtracking. This means that if a **node** has children, the next **node selected** is one of them. Otherwise we backtrack by following the unique path from the current node back to the root node (original problem) until we find a node (if any) with an unprocessed **child** .

The advantages of depth first search are:

1. Solving LPR for a **child** , given the optimal solution for a **parent** , just involves changing a single bound, and therefore, reoptimization by the dual simplex method should be very...

Programming Techniques and Data Structures

Ian Munro Editor

Planar Point Location Using Persistent Search Trees

NEIL SARNAK and ROBERT E. TARJAN

ABSTRACT: A classical problem in computational geometry is the planar point location problem. This problem calls for preprocessing a polygonal subdivision of the plane defined by n line segments so that, given a sequence of points, the polygon containing each point can be determined quickly on-line. Several ways of solving this problem in O(log n) query time and O(n) space are known, but they are all rather complicated. We propose a simple O(log n)-query-time, O(n)-space solution, using persistent search trees. A persistent search tree differs from an ordinary search tree in that after an insertion or deletion, the old version of the tree can still be accessed. We develop a persistent form of binary search tree that supports insertions and deletions in the present and queries in the past. The time per query or update is O(log m), where m is the total number of updates, and the space needed is O(1) per update. Our planar point location algorithm is an immediate application of this data structure. The structure also provides an alternative to Chazelle's "hive graph" structure, which has a variety of applications in geometric retrieval.

1. PLANAR POINT LOCATION

Let us consider a classical geometric retrieval problem. Suppose the Euclidian plane is subdivided into polygons by n line segments[1] that intersect only at their endpoints. (See Figure 1, p. 670.) Given such a polygonal subdivision and a sequence of query points in the plane, the *planar point location problem* is the problem of determining, for each query point, the polygon containing it. (For simplicity we shall assume that no query point lies on a line segment of the subdivision.) We require that the answers to the queries be produced on-line; that is, each query point must be located before the next one is known.

A solution to the point location problem consists of an algorithm that preprocesses the polygonal subdivision, building a data structure that facilitates location of individual query points. We measure the efficiency of such a solution by three parameters: the preprocessing time, the space required to store the data structure, and the time per query. Of these, the preprocessing time is generally the least important.

Many solutions to the point location problem have been proposed [10, 11, 13, 18, 22, 23, 32]. If binary decisions are used to locate the query points, $\Omega(\log n)$ time per query is necessary. Dobkin and Lipton [11] showed that this lower bound is tight, exhibiting a method with $O(\log n)$ query time needing $O(n^2)$ space and preprocessing time. The Dobkin-Lipton result raised the question of whether an $O(\log n)$ bound on query time can be achieved using only $O(n)$ space, which is optimal if the planar subdivision must be stored. Lipton and Tarjan [23] answered this question affirmatively by devising a

[1] We regard a line or half-line as being a line segment, and an infinite region whose boundary consists of a finite number of line segments as being a polygon.

complicated method based on the planar separator theorem [24].

More recent research has focused on providing a simpler algorithm with resource bounds the same as or close to those of the Lipton-Tarjan method. Algorithms with $O(\log n)$ query time using $O(n)$ space have been developed by Kirkpatrick [18], who used the fact that every planar graph has an independent set containing a fixed fraction of the vertices; by Edelsbrunner, Guibas, and Stolfi [13], who improved a method of Lee and Preparata [22] that uses the notion of separating chains; and by Cole [10], who noted that the Dobkin-Lipton approach reduces planar point location to a problem of storing and accessing a set of similar lists.

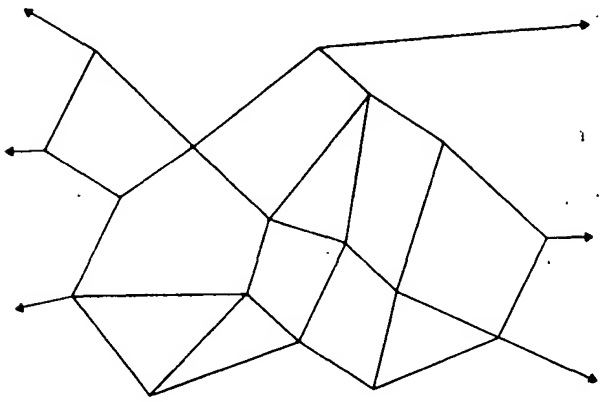

FIGURE 1. A Polygonal Subdivision. Arrows denote line segments going to infinity.

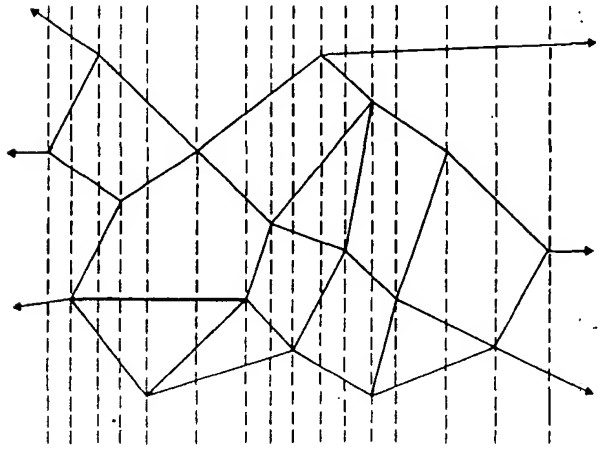

FIGURE 2. The Polygonal Subdivision of Figure 1 Divided into Slabs. The dashed lines are slab boundaries.

Cole's observation is the starting point for our work. Let us review the Dobkin-Lipton construction. Draw a vertical line through each vertex (intersection of line segments) in the planar subdivision. (See Figure 2.) This splits the plane into vertical slabs. The line segments of the subdivision intersecting a slab are totally ordered, from the bottom to the top of the slab. Associate with each line segment the polygon just above it. Now it is possible to locate a query point with two binary searches: the first, on the x-coordinate, locates the slab containing the point; the second, on the line segments intersecting the slab, locates the nearest line segment below the point, and hence determines the polygon containing the point. (By introducing a dummy line segment running from $(-\infty, -\infty)$ to $(\infty, -\infty)$, we can guarantee that below every point there is a line segment.) Since testing whether a point is above or below a line segment takes $O(1)$ time, a point query takes $O(\log n)$ time. Unfortunately, if we build a separate search structure (such as a binary search tree) for each slab, the worst-case space requirement is $\Theta(n^2)$, since $\Theta(n)$ line segments can intersect $\Theta(n)$ slabs.

We can reduce the space bound by noticing as Cole did that the sets of line segments intersecting contiguous slabs are similar. Think of the x-coordinate as time. Consider how the set of line segments intersecting the current slab changes as the time increases from $-\infty$ to $+\infty$. As the boundary from one slab to the next is crossed, certain segments are deleted from the set and other segments are inserted. Over the entire time range, there are $2n$ insertions and deletions, one insertion and one deletion per segment. (Think of line segments going to $-\infty$ in the x-coordinate as being inserted at time $-\infty$, and line segments going to $+\infty$ in the x-coordinate as being deleted at time $+\infty$.)

We have thus reduced the point location problem to the problem of storing a sorted set subject to insertions and deletions so that all past versions of the set, as well as the current version, can be accessed efficiently. In general we shall call a data structure *persistent* if the current version of the structure can be modified and all versions of the structure, past and present, can be accessed. Ordinary data structures, which do not support access in the past, we call *ephemeral*.

Cole solved the point location problem by devising a persistent representation of sorted sets that occupies $O(m)$ space and has $O(\log m)$ access time, where m is the total number of updates (insertions and deletions) starting from an empty set. However, his data structure has two drawbacks. First, his method is indirect, proceeding by way of an intermediate problem in which item substitutions but neither in-

sertions nor deletions are allowed. Second, the entire sequence of updates must be known in advance, making the data structure unusable in situations where the updates take place on-line. We shall propose a simpler data structure that overcomes these drawbacks.

Our main result, presented in Section 3, is a persistent form of binary search tree with an $O(\log m)$ worst-case access/insert/delete time and an amortized[2] space requirement of $O(1)$ per update. Our structure has neither of the drawbacks of Cole's. It provides a simple $O(n)$-space, $O(\log n)$-query-time point location algorithm. It can also replace Chazelle's "hive graph" [7], a rather complicated data structure with a variety of uses in geometric searching. Section 4 contains a brief discussion of these applications and some remarks about extensions and open problems. Some of the results presented here appear in preliminary form in Sarnak [34].

2. PERSISTENT SORTED SETS AND SEARCH TREES

We are now faced with a problem that is purely in the realm of data structures, the *persistent sorted set problem*. We wish to maintain a set of items that changes over time. The items have distinct *keys*, with the property that any collection of keys of items that are in the set simultaneously can be totally ordered. (The keys of two items that are not in the set at the same time need not be comparable.) Three operations on the set are allowed:

access(x, s, t): Find and return the item in set s at time t with greatest key less than or equal to x. If there is no such item, return a special *null* item.

insert(i, s, t): At time t, insert item i (with predefined key) into set s, assuming it is not already there. Item i remains in the set until it is explicitly deleted.

delete(i, s, t): At time t, delete item i from set s, assuming it is there.

Starting with an empty set, we wish to perform on-line a sequence of operations, including m updates (insertions and deletions), with the following property:

(*) Any update occurs at a time no earlier than any previous operation in the sequence. That is, updates are allowed only in the present.

The explicit time parameter t in the operations formalizes the notion of persistence. We break ties in operation time by order in the sequence of operations. Property (*) allows accesses to take place either in the present (after the most recent update) or in the past. In the usual ephemeral version of the sorted set problem, the time of an operation is implicit, corresponding to its position in the sequence of operations. An equivalent definition of the ephemeral problem is obtained by requiring the sequence of operations to have the following stronger property in place of (*): the operations in the sequence occur in nondecreasing order by time.

This problem and variants of it have been studied by many authors [8, 10, 12, 21, 27, 28, 31, 33, 36]. Dobkin and Munro [12] considered the problem of maintaining a persistent list subject to access, insertion, and deletion by list position. (The items in the list have positions 1 through n counting from the front to the back of the list.) The persistent list problem seems to be harder than the persistent sorted set problem. Dobkin and Munro proposed an off-line method (all updates occur in the sequence before all accesses) with $O((\log m)^2)$ access time using $O(m \log m)$ space. Overmars [31] proposed an on-line method for the persistent list problem with $O(\log m)$ access time using $O(m \log m)$ space. Overmars also studied the much easier version of the persistent sorted set problem in which an operation *access*(x, t) need only return an item if the set contains an item with key exactly equal to x. For this version, he developed an $O(m)$-space, $O(\log m)$-access-time on-line algorithm. Chazelle [8] devised an $O(m)$-space, $O((\log m)^2)$-access-time method for the off-line version of the original persistent sorted set problem. As discussed in Section 1, Cole [10] discovered an $O(m)$-space, $O(\log m)$-access-time off-line algorithm.

All these methods use data structures that are somewhat ad hoc and baroque. A more direct approach is to start with an ephemeral data structure for sorted sets or lists and make it persistent. This idea was pursued independently by Myers [27, 28], Krijnen and Meertens [21], Reps, Teitelbaum, and Demers [33], and Swart [36], who independently proposed essentially the same idea, which we shall call *path copying*. The resulting data structure can be used to represent both persistent sorted sets and persistent lists with an $O(\log m)$ time bound per operation and an $O(\log m)$ space bound per update.

In the remainder of this section we shall review binary search trees and how they can be made persistent using path copying. In Section 3 we propose a new method that uses space even more efficiently than path copying. It leads to a data structure for persistent sorted sets (but not persistent lists) that has bounds of $O(\log m)$ worst-case time per operation and $O(1)$ amortized space per update.

[2] By *amortized complexity* we mean the complexity of an operation averaged over a worst-case sequence of operations. For a full discussion on this concept, see Tarjan's survey paper [39].

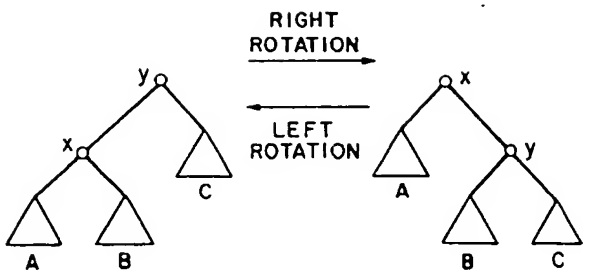

FIGURE 3. A Rotation in a Binary Tree. The tree can be a subtree of a larger tree.

A standard data structure for representing ephemeral sorted sets is the *binary search tree*. This is a binary tree[3] containing the items of the set in its nodes, one item per node, with the items arranged in *symmetric order*: if x is any node, the key of the item in x is greater than the keys of all items in its left subtree and less than the keys of all items in its right subtree. The symmetric-order item arrangement allows us to perform an access operation by starting at the tree root and searching down through the tree, along a path determined by comparisons of the query key with the keys of items in the tree: if the query key is equal to the key of the item in the current node, we terminate the access by returning the item in the current node; if it is less, we proceed to the left child of the current node; if it is greater, we proceed to the right child. Either the search terminates having found an item with key equal to the query key, or it runs off the bottom of the tree. In the latter case, we return the item in the node from which the search last went right; if there is no such node, we return null.

The time for an access operation in the worst case is proportional to the depth of the tree. If the tree is binary, its depth is at least $\lfloor \log n \rfloor + 1$, where n is the number of tree nodes. This bound is tight for *balanced binary trees*, which have depth $O(\log n)$ and insertion and deletion time bounds of $O(\log n)$ as well. There are many types of balanced trees, including *AVL* or *height-balanced trees* [1], *trees of bounded balance* or *weight-balanced trees* [29], and *red-black trees* [14]. In such trees balance is maintained by storing certain *balance information* in each node (of a kind that depends upon the type of tree) and rebalancing after an insertion or deletion by performing a series of *rotations* along the access path (the path from the root to the inserted or deleted item). A rotation (see Figure 3) is a local transformation that changes the depths of certain nodes, preserves symmetric order, and takes $O(1)$ time, assuming that a standard binary tree representation is used such as storing two pointers in each node, to its left and right children.

[3] See the books of Knuth [19] and Tarjan [37] for our tree terminology.

For definiteness, we shall concentrate on red-black trees, although our ideas apply to certain other kinds of balanced trees. In a red-black tree each node has a color, either *red* or *black*, subject to the following constraints:

(i) all missing (external) nodes are regarded as black;

(ii) all paths from the root to a missing node contain the same number of black nodes;

(iii) any red node, if it has a parent, has a black parent.

This definition is due to Guibas and Sedgewick [14]. Bayer [3] introduced these trees, calling them *symmetric binary B-trees*. Olivie [30] gave an equivalent definition (see [38]) and used the term *half-balanced* trees.

Updating red-black trees is especially efficient as compared to updating other kinds of balanced trees. Rebalancing after an insertion or deletion can be

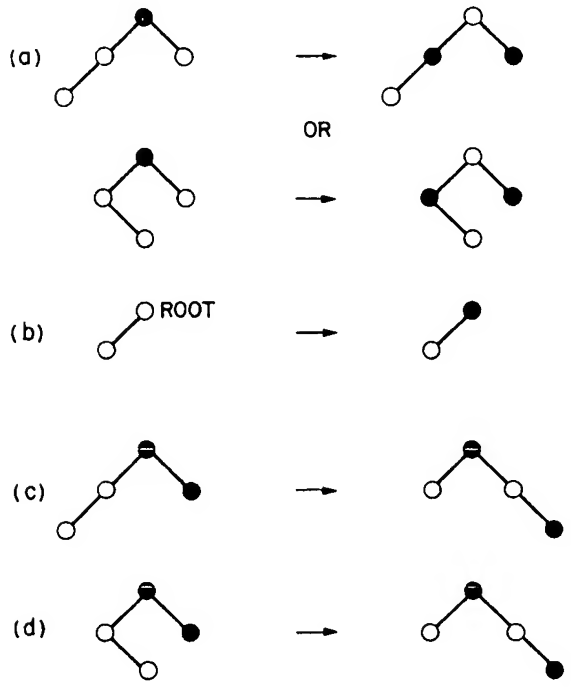

FIGURE 4. The Rebalancing Transformations in Red-Black Tree Insertion. Symmetric cases are omitted. Solid nodes are black; hollow nodes are red. All unshown children of red nodes are black. In cases (c) and (d) the bottommost black node can be missing.

done in $O(1)$ rotations and $O(\log n)$ color changes [38]. Furthermore the number of color changes per update is $O(1)$ in the amortized case [15, 16, 25]. Rebalancing is a bottom-up process. To perform an insertion, we proceed as in an access operation. At the place where the search runs off the bottom of the tree, we attach a new node containing the new item. We color this node red. This preserves the black constraint (ii) but may violate the red constraint (iii). If there are now two red nodes in a row the topmost of which has a red sibling, we color the topmost red node, its red sibling black, and their common parent (which must be black) red. (See Figure 4a.) This may produce a new violation of the red constraint. We repeat the transformation of Figure 4a, moving the violation up the tree, until this transformation no longer applies. If there is still a violation, we apply the appropriate transformation among those in Figure 4b, c, and d to eliminate the violation. This terminates the insertion. The only rotations are in the terminal cases: Figure 4c takes one rotation and Figure 4d takes two.

A deletion is similar. We first search for the item to be deleted. If it is in a node with a left child, we swap the item with its predecessor (in symmetric order), which we find by taking a left branch and then right branches until reaching a node with no right child. Now the item to be deleted is in a node with at most one child. We delete this node and replace it with its child (if any). This does not affect the red constraint but will violate the black constraint if the deleted node was black. If there is a violation, the replacing node (which may be missing) is *short*; paths down from it contain one fewer black node than paths down from its sibling. We bubble the shortness up the tree by repeating the recoloring transformation of Figure 5a until it no longer applies. Then we perform the transformation of Figure 5b if it applies, followed if necessary by one application of Figure 5c, d, or e. The maximum number of rotations needed is three.

Let us now consider how to make red-black trees persistent. We need a way to retain the old version of the tree when a new version is created by an update. We can of course copy the entire tree each time an update occurs, but this takes $O(n)$ time and space per update. The idea of Myers [27, 28], Krijnen and Meertens [21], Reps, Teitelbaum, and Demers [33], and Swart [36] is to copy only the nodes in which changes are made. Any node that contains a pointer to a node that is copied must itself be copied. Assuming that every node contains pointers only to its children, this means that copying one node requires copying the entire path to the node from the root of the tree. Thus we shall call this method *path copying*. The effect of this method is to create a set of search trees, one per update, having different roots but sharing common subtrees. Since node colors are needed only for update operations, all of which take place in the most recent version of the tree, we need not copy a node when its color changes; we merely overwrite the old color. This saves a constant factor in space. (See Figure 6, p. 674.)

The time and space per insertion or deletion in a persistent red-black tree is $O(\log n)$ since such an operation changes only nodes along a single path in the tree. If the update times are arbitrary real numbers, we must build an auxiliary structure to facilitate access to the appropriate root when searching in the past. An array of pointers to the roots, ordered by time of creation, suffices. We can use binary search in this array to access the appropriate root. This increases the time per access from $O(\log n)$ to $O(\log m)$. If we use exponential search, the time to perform an access in the tth version of the tree can be reduced to $O(\log n + \log t)$: we examine the first, second, fourth, . . . , $2^{\lfloor \log t \rfloor}$th root until finding one created after the desired search time; then we use

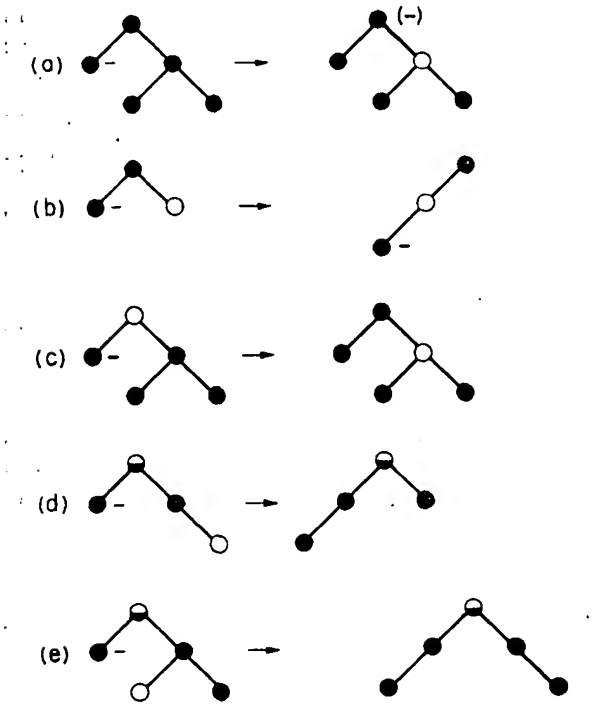

FIGURE 5. The Rebalancing Transformation in Red-Black Tree Deletion. The two ambiguous (half-solid) nodes in (d) have the same color, as do the two in (e). Minus signs denote short nodes. In (a), the top node after the transformation is short unless it is the root.

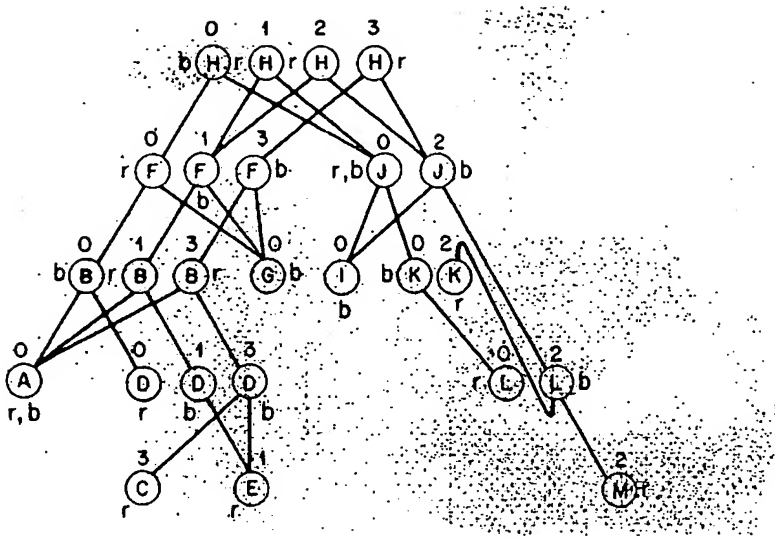

FIGURE 6. A Persistent Red-Black Tree With Path Copying. The initial tree, existing at time 0, contains A, B, D, F, G, H, I, J, K. Item E is inserted at time 1, item M at time 2, and item C at time 3. The nodes are labeled by their colors, *r* for red, *b* for black. The nodes are also labeled by their time of creation. All edges exit the bottoms of nodes and enter the tops.

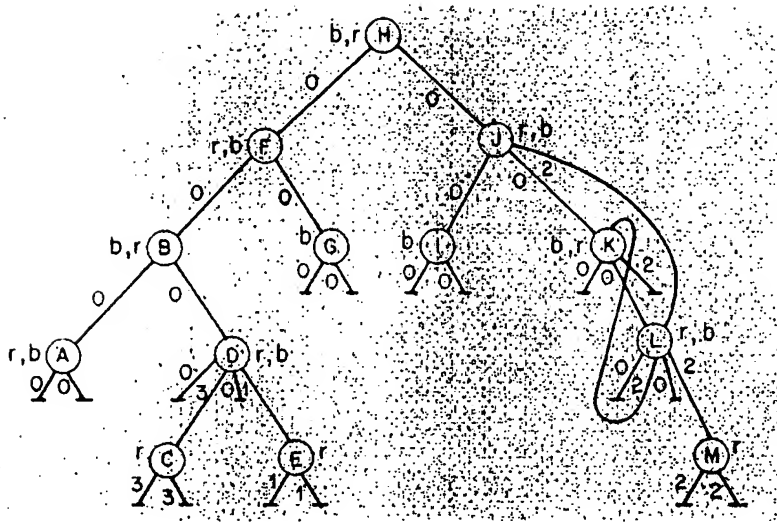

FIGURE 7. A Persistent Red-Black Tree With No Node Copying. The initial tree and insertions are as in Figure 6. The edges are labeled with their time of creation, the nodes are labeled with their colors. Connections to horizontal lines denote null pointers.

binary search on the roots from 1 through $2\lceil \log t \rceil$ (numbered in creation order). The same kind of search starting from the most recently created root and proceeding to earlier roots gives an access time of $O(\log n + \log(m - t))$. If the update times are the integers 1 through m, we can use direct access into the root array to provide $O(1)$-time access to the appropriate root, and the total time for an access operation is only $O(\log n)$.

As Swart noted, path copying works on any kind of balanced tree, not just on red-black trees. Myers used AVL trees, Krijnen and Meertens used B-trees, and Reps, Teitelbaum, and Demers used 2,3 trees. Path copying is also quite versatile in the applications it supports. By storing in each node the size of the subtree rooted there, we can obtain an implementation of persistent lists (in which access is by list position rather than by key). We also have the ability to update *any* version, rather than just the current one, provided that an update is assumed to create an entirely new version, independent of all other versions. In order to have this more general kind of updating, we must copy a node when its balance information changes as well as when one of its pointers changes, but this increases the time and space needed for updates by only a constant factor.

3. SPACE-EFFICIENT PERSISTENT SEARCH TREES

A major drawback of the path copying method is its nonlinear space usage. In this section we shall propose a method that needs only linear space. We shall

use the fact that old balance information need not be saved, although this is not essential. Our approach is to avoid copying the entire access path each time an update occurs. That this approach might work is suggested by the observation that in an ephemeral red-black tree, only $O(1)$ pointer changes are needed per update.

Suppose we implement persistent red-black trees without any node copying, by allowing nodes to become arbitrarily "fat": each time we want to change a pointer, we store the new pointer in the node, along with a time stamp indicating when the change occurred and a bit that indicates whether the new pointer is a left or right pointer. (This bit is actually redundant, since we can determine whether a pointer is left or right by comparing the key of the item in the node containing the pointer to that of the item in the node indicated by the pointer.) When a node color is changed we overwrite the old color. (See Figure 7.)

With this approach an insertion or deletion in a persistent red-black tree takes only $O(1)$ space, since an insertion creates only one new node and either kind of update causes only $O(1)$ pointer changes. The drawback of the method is its time penalty: since a node can contain an arbitrary number of left or right pointers, deciding which one to follow during a search is not a constant-time operation. If we use binary search by time stamp to decide which pointer to follow, choosing the correct pointer takes $O(\log m)$ time, and the time for an access, insertion, or deletion is $O((\log n)(\log m))$.

We can eliminate this time penalty by introducing limited node copying. We allow each node to hold k pointers in addition to its original two. We choose k to be a small positive constant; $k = 1$ will do. When attempting to add a pointer to a node, if there is no empty slot for a new pointer, we copy the node, setting the initial left and right pointers of the copy to their latest values. (Thus the new node has k empty slots.) We must also store a pointer to the copy in the latest parent of the copied node. If the parent has no free slot, it, too, is copied. Thus copying proliferates through successive ancestors until the root is copied or a node with a free slot is reached. (See Figure 8.)

Searching the resulting data structure is quite easy: when arriving at a node, we determine which pointer to follow by examining the key to decide whether to branch left or right and examining the time stamps of the extra pointers to select among multiple left or multiple right pointers. (We follow the pointer with the latest time stamp no greater than the search time if there is one, or else the initial pointer.) As noted in Section 2, if the update times are arbitrary real numbers we must build an auxiliary array to guide access operations to the proper roots. This makes the time for an access operation $O(\log m)$, whereas the time for an update operation is $O(\log n)$. However, in practice the number of roots is likely to be much smaller than m, since a root will be duplicated relatively infrequently. If the update times are consecutive integers, the auxiliary array provides $O(1)$-time access to the roots.

It remains for us to analyze the space used by the data structure. As with path copying, a single update operation using limited node copying can result in $O(\log n)$ new nodes. However, amortized over a sequence of updates, there are only $O(1)$ nodes copied per update, implying an $O(n)$ space bound for the

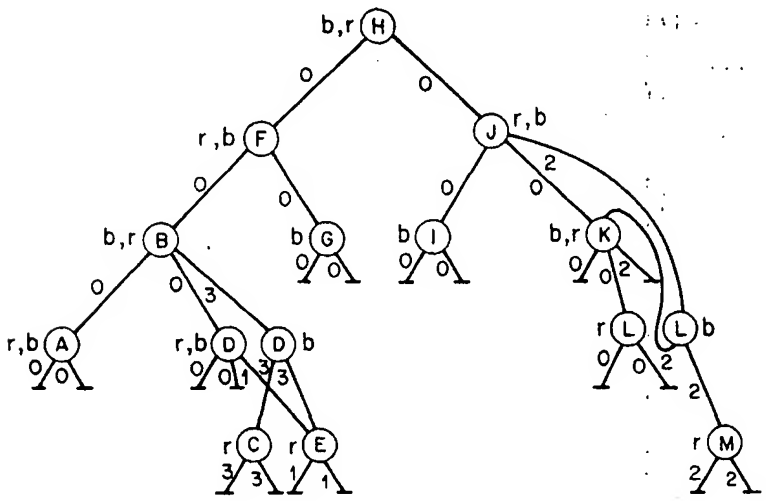

FIGURE 8. A Persistent Red-Black Tree With Limited Node Copying Assuming Each Node Can Hold One Extra Pointer. The initial tree and insertions are as in Figure 6. The labeling is as in Figure 7.

data structure. To obtain the amortized space bound we need some definitions. We partition the nodes of the data structure into two classes, *live* and *dead*. The live nodes are those reachable from the latest tree root by following pointers valid at the current time (the time of the most recent update). The live nodes form the current version of the search tree. As the current time increases, the node partition changes: live nodes can become dead but not vice-versa. All nodes dead at a given time are not affected by any later update.

Our analysis uses the potential paradigm [39]. We define the *potential* of the data structure to be the number of live nodes minus $1/k$ times the number of free slots in live nodes. We define the *amortized space cost* of an update operation to be the actual number of nodes it creates plus the net increase in potential it causes. With these definitions, the actual number of nodes created by a sequence of updates is bounded by the sum over all updates of the amortized space cost plus the net decrease in potential over the sequence. If we start with an empty data structure, the initial potential is zero, and since the potential is always nonnegative the total amortized space cost is an upper bound on the actual number of nodes created.

The definition of potential is such that copying a node has an amortized space cost of zero, since a live node with no free slots becomes dead and a new live node with k free slots is created, for a net decrease in potential of one, balancing the one new node created. Storing a new pointer in a node has an amortized space cost of $1/k$. The creation of a new node during an insertion has an amortized space cost of one. Since an insertion or deletion requires storing $O(1)$ new pointers not counting node copying, the amortized space cost of an update is $O(1)$. A more careful count shows that an insertion has an amortized space cost of at most $1 + 6/k$; a deletion, at most $7/k$. In the special case of $k = 1$, the amortized space cost per update is slightly less than indicated by these bounds: at most six for an insertion or deletion.

The choice $k = 1$ is probably the most convenient in practice and is certainly the easiest to implement. However, choosing a larger value of k may reduce the space needed by the data structure, since although the space per node increases, the number of node copyings decreases. The best choice of k depends on the exact way nodes are stored in memory and on the average (as opposed to worst-case) number of new pointers created by updates. Nevertheless, we shall give a simplified analysis based on the amortized bounds derived above. Suppose that memory is divided into words, each of which is large enough to hold an item, a time stamp, or a pointer. We shall ignore the space needed to store node colors and the types of extra pointers (left or right); as noted above the latter information is redundant and the color of a node can if necessary be encoded by swapping or not swapping the original left and right pointers in a node. Under these assumptions a node requires $2k + 3$ words of memory, and the amortized space cost in words per update is at most $(2k + 3)(1 + 6/k) = 2k + 18/k + 15$. This is minimized at 27 words per update for $k = 3$. This choice is only marginally better than the 30 words per update (six nodes of five words each) needed for $k = 1$. Both these estimates are probably much larger than the expected values.

Limited node copying applied to red-black trees provides a linear-space representation of persistent sorted sets but not of persistent lists, because to represent lists we must maintain subtree sizes for all versions, and each update causes $O(\log n)$ subtree sizes to change. Limited node copying becomes similar to path copying in this case, and the space bound per update is $O(\log n)$. Our data structure does, however, support operations on persistent sorted sets in addition to those defined in Section 2. In particular, the following three operations are easy to handle:

access range (x, y, s, t): Find and return all items in set s at time t with key between x and y (inclusive).

join (s_1, s_2, t): At time t, combine sets s_1 and s_2 into a single set, named s_1. Set s_2 becomes empty at time t. This operation requires that at time t all items in s_1 have keys less than those of all items in s_2. Time t can be any time greater than or equal to the time of the most recent update.

split (s_1, s_2, x, t): At time t, split s_1 into two sets: a new version of s_1, containing all items with key less than or equal to x, and s_2, containing all items with key greater than x. Time t can be any time greater than or equal to the time of the most recent update.

We shall discuss how to implement these operations on ephemeral red-black trees; the extensions to persistent trees are straightforward. To perform *access range* (x, y, s, t), we proceed as in *access* (x, s, t), thereby locating the node e containing the item with smallest key no less than x. Then we visit the tree nodes starting from e in symmetric order, stopping when we reach one containing an item with key exceeding y. In an ephemeral tree, the time for such a query is $O(k + \log n)$, where k is the number of

items returned. In a persistent tree the time is $O(k + \log m)$ assuming update times are arbitrary real numbers.

To discuss joining and splitting, we need the concept of the *rank* $r(e)$ of a node e, defined to be the number of black nodes on any path from e down to a missing node. We can compute the rank of a node in time proportional to the rank by walking down from the node along any path. (Instead of comparing ranks from scratch, we can store with each tree root its rank, and then compute ranks on the way down the tree along any search path.) Consider a join of sets s_1 and s_2. To perform the join, we delete the item, say i, of smallest key in set s_2. We compute the ranks r_1 and r_2 of the roots of the trees T_1 and T_2 representing s_1 and the new s_2, respectively. Assume $r_1 \geq r_2$. (The case $r_1 \leq r_2$ is symmetric.) If $r_1 = r_2$, we create a new black node containing i and make the roots of T_1 and T_2 its children. If $r_1 = r_2 + 1$ and the root of T_2 is red, we color it black and proceed as in the case of $r_1 = r_2$. Otherwise, we color the root of T_2 black if it is red and locate the node e along the right path[4] of T_1 of one higher rank than the root of T_2. We create a new red node containing i, which becomes the new right child of e; its left child is the old right child of e and its right child is the root of T_2. This may create a violation of the red constraint, which we eliminate as in insertion. The total time taken by the join is $O(\log n)$, where n is the size of the new tree. Since only one new node is created and $O(1)$ pointer changes are made, the amortized space bound in the persistent version is $O(1)$. Note that once i is deleted from T_2, and r_1 and r_2 are computed, the time for the rest of the join is $O(r_1 - r_2 + 1)$. Furthermore, the rank of the root of the new tree is either r_1 or $r_1 + 1$.

We implement splitting using repeated joining. The easiest way to describe the algorithm is recursively. Suppose we have a procedure *join3* whose effect is as follows:

join3 (e, f, g): Let e, f, g be nodes such that e and g are the roots of red-black trees T_1 and T_2, respectively, satisfying the condition that all items in T_1 have keys smaller than that of the item in f and all items in T_2 have keys greater than that of the item in f. Combine T_1, f, and T_2 into a single tree whose root has rank $\max\{r(e), r(g)\}$ or $\max\{r(e), r(g)\} + 1$, and return the root of the new tree.

We implement *join3* in the same way as the second half of a binary join; the time it requires is $O(|r(e) - r(g) + 1|)$. Using *join3*, we can implement a procedure *split*(e, x), whose input is the root e of a red-black T and a key x, and whose output is a pair (f, g) such that f and g are the roots of the trees formed when T is split at x. Let *left*(e) and *right*(e) be the left and right children of node e, respectively. To perform *split*(e, x), we test whether the key of the item in e is less than, or equal to x. If so, we perform *split*(*right*$(e), x)$, returning (h, g). Then we compute $f =$ *join3* (*left*(e), e, h) and return (f, g). The case of x less than the key of the item in e is symmetric.

The splitting algorithm has a running time of $O(\log n)$, because the multiple joins that take place have running times that form a telescoping sum, summing to $O(\log n)$. (See, for example [2].) The persistent version has an amortized space bound of $O(\log n)$. We can reduce this amortized space bound to $O(\log \min\{k, n - k\})$, where k and $n - k$ are the sizes of the trees resulting from the split, by modifying the splitting algorithm slightly. To split a tree T with root r at key x, we follow the search path for x until it changes direction. Suppose the first change of direction is from right to left (the opposite case is symmetric), and let e be the node from which we branch left. (Node e is the last node along the search path that is on the right path of T.) We break the link connecting e to its parent f and perform *split*(e, x) (as implemented above) returning (g, h). We replace f as the right child of its parent by *left*(f), repairing the possible violation of the color constraints as in the deletion algorithm. Finally, we return the pair (*join3*(r, f, g), h). The time bound is still $O(\log n)$. The amortized space bound of $O(\log \min\{k, n - k\})$ for the persistent version follows from two facts: (i) node e has rank $O(\log \min\{k, n - k\})$ in the original tree; (ii) restoring the color constraints after replacing node f by its left child takes only $O(1)$ pointer changes.

Maintaining more than one persistent sorted set (as one must do if joins and splits are allowed) requires the maintenance of an auxiliary structure for each set to facilitate access to the appropriate root when searching. If multiple arrays are hard to use as auxiliary structures because of the problem of allocating storage for them, search trees can be used instead. The trees can be either ordinary balanced trees or some other kind, such as finger search trees[5] or self-adjusting trees [35]. Depending on the choice of structure, the time to access the appropriate root is $O(\log m)$ or faster.

We conclude this section with a few remarks about the generality of our $O(1)$ amortized space bound for insertion, deletion, and join. What makes the analysis work is that red-black trees need only

[4] The *right path* of a binary tree is the path from the root through right children to a missing node. The *left path* is defined similarly.

[5] A *finger search tree* is a search tree augmented with a few pointers to favored nodes, called *fingers*. Access and update operations in the vicinity of fingers are especially efficient [6, 16, 17, 20, 41].

$O(1)$ pointer changes per update. This bound happens to be worst-case, but for our purpose an amortized bound would do as well, since the resulting space bound is amortized anyway. This means that any kind of balanced tree with $O(1)$ amortized structural update times can be used in place of red-black trees. Examples include red-black trees with top-down instead of bottom-up updating [26], weight-balanced trees [5], and "weak" or "hysterical" B-trees [15, 16, 25]. We also have the option of storing the items in the external nodes of the tree instead of in the internal nodes (if we store appropriate keys in the internal nodes to guide searches).

4. APPLICATIONS AND EXTENSIONS

We have proposed a data structure for representing persistent sorted sets. Our structure has $O(\log m)$ access time, $O(\log n)$ update time, and needs $O(1)$ amortized space per update starting from an empty set. Here n is the current set size and m is the total number of updates. Our resource bounds match those of Cole [10], but our data structure is on-line and is simple enough to have potential practical applications.

As discussed in Section 1, our structure provides an efficient solution to the planar point location problem. For a planar subdivision of n line segments, the preprocessing time necessary to build the data structure is $O(n \log n)$, the space needed is $O(n)$, and the query time is $O(\log n)$. Although these bounds have been obtained by others [10, 14, 18, 23], our method is simple enough to be useful in practice as well as efficient in theory. The methods of Kirkpatrick [18] and Edelsbrunner, Guibas, and Stolfi [13], when combined with a new linear-time algorithm for triangulating simple polygons [40], need $O(n)$ preprocessing time rather than $O(n \log n)$. Whether this reduction is important depends on the application. It is open whether some variant of our method has $O(n)$ preprocessing time.

Our structure also supports a generalization of the planar point location problem in which the queries are of the following form: given a vertical line segment, report all polygons the segment intersects. Such a query is equivalent to an access range operation on the corresponding persistent sorted set and thus takes $O(\log n + k)$ time where k is the number of reported polygons. This bound has also been obtained by Chazelle [7], but only by using a complicated data structure, the *hive graph*, which is built as an extension to a data structure for the planar point location problem. Our structure solves both problems at once.

Chazelle gives a number of applications of hive graphs to geometric retrieval problems; for each of these, our structure provides a simpler solution. As an example, given a collection of line segments in the plane with i crossings, we can in $O((n + i)\log n)$ time construct a data structure of size $O(n + i)$ that, given a vertical query segment, will allow us to report all data line segments the query segment crosses in $O(\log n + k)$ time, where k is the number of reported segments. Cole [18] gives several other applications to which our structure applies.

We have obtained several extensions to the result presented here, which we shall discuss in detail in a future paper. The limited node copying technique generalizes to show that any ephemeral linked data structure, provided its nodes have constant in-degree as well as constant out-degree, can be made persistent at an amortized space cost of $O(1)$ per structural change and an additive $O(\log m)$ time penalty per access. Whereas limited node copying as discussed in the present paper resembles node-splitting in B-trees, the generalized technique resembles the "fractional cascading" idea of Chazelle and Guibas [9]. Among other applications, the generalized technique allows the addition of extra pointers, such as parent pointers and level links [6], to persistent red-black trees.

Our implementation of persistent search trees, although more space-efficient than the path copying method, is not as versatile. For example, path copying provides a representation for persistent lists as well as persistent sorted sets. For the list application, limited node copying is equivalent to path copying because the size information necessary for access by position must be updated all the way along an access path after any insertion or deletion, causing $\Theta(\log n)$ space usage per update. As noted in Section 2, path copying also provides the ability to update *any* version, rather than just the current one. Adding additional pointers, such as parent pointers, to the resulting data structure seems difficult. Nevertheless, path copying can be extended to finger search trees, reducing the space usage for updates in the vicinity of fingers.

There are many open problems concerning geometric retrieval problems and persistent data structures. Perhaps one of the most interesting is how to make our planar point location algorithm, or any such algorithm, dynamic, so that line segments can be inserted and deleted on-line. The dynamization techniques of Bentley and Saxe [4] provide a way to handle insertions while preserving the $O(1)$ space bound. However, the access and insertion time becomes $O(\log n)^2)$. Deletion seems to be harder to handle. An even more challenging problem is to find a persistent representation for a dynamically changing planar subdivision. A good data structure for this purpose would have many applications in computational geometry [10].

REFERENCES

1. Adelson-Velškii, G.M., and Landis, E.M. An algorithm for the organization of information. *Soviet Math. Dokl. 3*, 5 (Sept. 1962), 1259-1262.
2. Aho, A.V., Hopcroft, J.E., and Ullman, J.D. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, Mass., 1974.
3. Bayer, R. Symmetric binary B-trees: Data structure and maintenance algorithms. *Acta Inform. 1*, 4 (Nov. 1972), 290-306.
4. Bentley, J.L., and Saxe, J.B. Decomposable searching problems I: Static-to-dynamic transformation. *J. Algorithms 1*, 4 (Dec. 1980), 301-358.
5. Blum, N., and Mehlhorn, K. On the average number of rebalancing operations in weight-balanced trees. A-78/06, Fachbereich Angewandte Mathematik und Informatik, Universität des Saarlandes, Saarbrücken, West Germany, 1978.
6. Brown, M.R., and Tarjan, R.E. Design and analysis of data structures for representing sorted lists. *SIAM J. Comput. 9*, 3 (Aug. 1980), 594-614.
7. Chazelle, B. Filtering search: A new approach to query-answering. In *Proceedings of 24th Annual IEEE Symposium on Foundations of Computer Science*. (Tucson, Ariz., Nov. 7-9, 1983), 122-132.
8. Chazelle, B. How to search in history. *Inform. Control*, to appear.
9. Chazelle, B., and Guibas, L.J. Fractional cascading: A data structuring technique with geometric applications. In *Automata, Languages, and Programming, 12th Colloquium*, (Napflion, Greece, July 15-18, 1985): *Lecture Notes in Computer Science 194*. Wilfried Bauer, Ed., Springer-Verlag, Berlin, 1985, 90-100.
10. Cole, R. Searching and storing similar lists. *J. Algorithms*, to appear.
11. Dobkin, D., and Lipton, R.J. Multidimensional search problems. *SIAM J. Comput. 5*, 2 (June 1976), 181-186.
12. Dobkin, D.P., and Munro, J.I. Efficient uses of the past. In *Proceedings of 21st Annual IEEE Symposium on Foundations of Computer Science*, (Syracuse, N.Y., Oct. 13-16, 1980), 200-206.
13. Edelsbrunner, H., Guibas, L.J., and Stolfi, J. Optimal point location in a monotone subdivision. Rep. 2, Digital Systems Research Center, Palo Alto, Calif., 1984.
14. Guibas, L.J., and Sedgewick, R. A dichromatic framework for balanced trees. In *Proceedings of 19th Annual IEEE Symposium on Foundations of Computer Science*, (Ann Arbor, Mich., Oct. 16-18, 1978), 8-21.
15. Huddleston, S., and Mehlhorn, K. Robust balancing in B-trees. *Lecture Notes in Computer Science 104*, Springer-Verlag, Berlin, West Germany (1981), 234-244.
16. Huddleston, S., and Mehlhorn, K. A new data structure for representing sorted lists. *Acta Inform. 17*, 2 (June 1982), 157-184.
17. Huddleston, S. An efficient scheme for fast local updates in linear lists. Department of Information and Computer Science, Univ. of California, Irvine, 1981.
18. Kirkpatrick, D. Optimal search in planar subdivisions. *SIAM J. Comput. 12*, 1 (Feb. 1983), 28-35.
19. Knuth, D.E. *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*. 2d. ed., Addison-Wesley, Reading, Mass., 1973.
20. Kosaraju, S.R. Localized search in sorted lists. In *Proceedings of 13th Annual ACM Symposium on Theory of Computing*, (Milwaukee, Wis., May 11-13, 1981), 62-69.
21. Krijnen, T., and Meertens, L.G.L.T. Making B-trees work for *B*. IW 219/83, The Mathematical Centre, Amsterdam, The Netherlands, 1983.
22. Lee, D.T., and Preparata, F.P. Location of a point in a planar subdivision and its applications. *SIAM J. Comput. 6*, 3 (Sept. 1977), 594-606.
23. Lipton, R.J., and Tarjan, R.E. Applications of a planar separator theorem. In *Proceedings of 18th Annual IEEE Symposium on Foundations of Computer Science*, (Providence, R.I., Oct. 11-Nov. 2, 1977), 162-170.
24. Lipton, R.J., and Tarjan, R.E. A separator theorem for planar graphs. *SIAM J. Appl. Math. 36*, 2 (Apr. 1979), 177-189.
25. Maier, D., and Salveter, S.C. Hysterical B-trees. *Inform. Process. Lett. 12*, 4 (Aug. 13, 1981), 199-202.
26. Mehlhorn, K. *Data Structures and Algorithms 1: Sorting and Searching*. Springer-Verlag, Berlin, 1984.
27. Myers, E. W. AVL dags. Tech. Rep. 82-9, Department of Computer Science, Univ. of Arizona, Tucson, Ariz., 1982.
28. Myers, E.W. Efficient applicative data types. In *Conference Record Eleventh Annual ACM Symposium on Principles of Programming Languages*, (Salt Lake City, Utah, Jan. 15-18, 1984), 66-75.
29. Nievergelt, J., and Reingold, E.M. Binary search trees of bounded balance. *SIAM J. Comput. 2*, 1 (Mar. 1973), 33-43.
30. Olivié, H. A new class of balanced search trees: Half-balanced binary search trees. *RAIRO Informatique Théoretique*, 16 (1982), 51-71.
31. Overmars, M.H. Searching in the past I. *Inform. Control*, to appear.
32. Preparata, F.P. A new approach to planar point location. *SIAM J. Comput. 10*, 3 (Aug. 1981), 473-482.
33. Reps, T., Teitelbaum, T., and Demers, A. Incremental context-dependent analysis for language-based editors. *ACM Trans. Prog. Lang. Syst.*, 5 (1983), 449-477.
34. Sarnak, N. Persistent data structures. Ph.D. dissertation, Department of Computer Science, New York University, New York, to appear.
35. Sleator, D.D., and Tarjan, R.E. Self-adjusting binary search trees. *J. ACM 32*, 3 (July 1985), 652-686.
36. Swart, G. Efficient algorithms for computing geometric intersections. Tech. Rep. #85-01-02, Department of Computer Science, Univ. of Washington, Seattle, 1985.
37. Tarjan, R.E. *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, Pa., 1983.
38. Tarjan, R.E. Updating a balanced search tree in *O*(1) rotations. *Inform. Process. Lett. 16*, 5 (June 1983), 253-257.
39. Tarjan, R.E. Amortized computational complexity. *SIAM J. Alg. Disc. Meth. 6*, 2 (Apr. 1985), 306-318.
40. Tarjan, R.E., and van Wyk, C.J. A linear-time algorithm for triangulating simple polygons. In *Proceedings of 18th Annual ACM Symposium on Theory of Computing*. (Berkeley, Calif., May 28-30, 1985), to appear.
41. Tsakalidis, A.K. AVL-trees for localized search. *Lecture Notes in Computer Science 172*, Springer-Verlag, Berlin, West Germany (1984), 473-485.

CR Categories and Subject Descriptors: E.1[Data]: Data Structures—*graphs, lists, trees*; E.2[Data]: Data Storage Representations—*linked representations*; F.2.2[Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems—*computations on discrete structures, geometrical problems and computations*; I.3.5[Computer Graphics]:
General Terms: Algorithms, Theory
Additional Key Words and Phrases: *post-office problem, planar point location, persistent data structure, search tree*

Received 8/85; revised 10/85; accepted 2/86

Authors' Present Addresses: Neil Sarnak, IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598. Robert E. Tarjan, Computer Science Department, Princeton University, Princeton, NJ 08544, and AT&T Bell Laboratories, Murray Hill, NJ 07974.

CORRIGENDUM

David S. Scott and S. Sitharama Iyengar, TID-A translation invariant data structure for storing images. *Commun. ACM 29*, 5(May 1986), 418-429.

Page 425, left column, paragraph 1, sentences 6 and 7 should read:

Maximal square characterization of Figure 9a using TID structure is described in Table II. Table III summarizes the best, worst, and average performance for the various locations.